Embedding F-Script into Cocoa Applications

Philippe Mougin (pmougin@acm.org)

Révision : Décembre 2005.

1. Introduction

Comme nous l'avons vu dans les précédents articles de cette série, F-Script peut être utilisé comme une application stand-alone, qui charge dynamiquement vos classes Objective-C et vous permet d'y accéder de manière interactive ou via des scripts. Dans cet article, nous allons explorer la possibilité inverse : inclure F-Script au sein de vos propres applications.

L'ensemble des fonctionnalités de F-Script est disponible via des composants intégrables au sein d'applications. Cela permet en autre de :

- Programmer tout ou partie d'une application en utilisant le langage F-Script.
- Proposer un environnement de scripting au sein de vos applications, permettant aux utilisateurs de manipuler vos objets métiers et d'automatiser des traitements.
- Interfacer F-Script à d'autres outils.

F-Script et Cocoa partageant le même modèle objet, l'intégration est facile et poussée.

2. Les principales classes de l'API d'intégration F-Script

L'intégration du run-time F-Script à une application se fait via un certain nombre de classes Objective-C livrées avec F-Script. Nous présentons dans le tableau ci-dessous les principales caractéristiques de ces classes dans le domaine de l'intégration.

Nom	Rôle	Principales Fonctionnalités
FSInterpreter	Chaque instance de cette classe représente un interpréteur F-Script complet (workspace inclus). Une application peut instancier un nombre quelconque d'interpréteurs.	• Exécuter le code F-Script contenu dans une NSString. • Rechercher l'objet associé à un certain nom [traduction: identifier] dans le workspace. • Assigner un objet à un nom dans le workspace. • Connaître les noms définis dans le workspace.
FSInterpreterResult	Représente le résultat d'une exécution de code F-Script par un FSInterpreter.	♦ Connaître le statut de l'exécution du code F-Script : erreur de syntaxe, erreur d'exécution ou succès. ♦ En cas d'erreur, récupérer les informations concernant cette erreur : human readable description, localisation dans le code, call-stack des blocks. ♦ En cas de succès, récupérer le résultat de l'exécution.
FSInterpreterView	Un composant graphique, sous classe de NSView, offrant une interface ligne de commande [traduction : command line interface] à F-Script. Ce composant est notamment utilisé par l'application « fs » pour son interface graphique. Un FSInterpreterVew contient son propre interpréteur et workspace F-Script et peut donc être utilisé directement sans autre besoin de configuration.	♦ S'intégrer dans une fenêtre Cocoa. ♦ Gérer l'interaction utilisateur en mode ligne de commande. Nombreuses fonctions : copier-coller, historique des commandes, choix des fonts, signalement graphique des erreurs etc. ♦ Insérer une commande F-Script de manière programmatique. ♦ Communiquer un message à l'utilisateur. ♦ Récupérer l'interpréteur associé (une instance de FSInterpreter).
System	Permets l'accès (depuis Objective-C comme depuis F-Script) à de nombreux services d'un interpréteur F-Script. Le run-time crée une instance de System par interpréteur et l'associe au nom « sys » dans le workspace de l'interpréteur.	♦ Fabriquer dans le workspace associé un objet Block à partir d'une NSString contenant du code F-Script (prend en compte le cas d'erreurs de syntaxe dans le code). ♦ Ouvrir un browser d'objet.[Traduction : graphical F-Script object browser]. ♦ Sauvegarder/recharger un workspace. ♦ Obtenir la liste de l'ensemble des variables définies dans le workspace associé.
Block	Un block représente un script (un bout de code F-Script pouvant avoir des arguments et des variables locales).	• S'exécuter (levée d'exception en cas d'erreur). • Gérer une interface graphique d'édition de code.
FSNSString	Une catégorie de NSString.	• Fabriquer un Block à partir du code F-Script représenté par l'objet NSString (prend en compte le cas d'erreurs de syntaxe dans le code).

Un programme souhaitant utiliser ces APIs doit utiliser le framework FScript.framework (dans Project Builder, selectionner "Add Framework..." dans le menu "project" et ajouter FScript.framework à votre projet) et utiliser la directive d'import suivante: `#import <FScript/Fscript.h>`

Afin d'assurer le chargement du framework F-Script à l'exécution, vous pouvez le placer dans l'un des répertoires standard réservés aux frameworks sous Mac OS X:
~/Library/Frameworks (où ~ représente votre home directory)
/Library/Frameworks
/Network/Library/Frameworks

Vous pouvez aussi packager le framework F-Script avec votre application.

3. Utilisation de la méthode asBlock

Cette méthode offre le moyen le plus simple de créer et d'exécuter du code F-Script depuis Objective-C. Invoquée sur une NSString contenant un block F-Script, elle génère et retourne un objet de classe Block. Celui-ci peut alors être exécuté. On peut lui passer des paramètres et récupérer le résultat de l'exécution. On remarquera que cette technique permet d'unifier l'utilisation des Blocks puisque, après leur création, ils sont manipulés de la même manière depuis F-Script et Objective-C.

Exemple 1: hello world

Voici comment exécuter depuis Objective-C un programme "hello world" écrit en F-Script (le code F-Script est coloré en violet) :

```
// Crée une string contenant le code F-Script
NSString *fscriptCode = @"[sys log:'hello world']";

// Crée un object Block à partir de la string
Block *block = [fscriptCode asBlock];

// Exécute le block
[block value];
```

Il est possible de combiner ces instructions en une seule :

```
[[@"[sys log:'hello world']" asBlock] value]
```

Note : en Objective-C, les brackets (i.e. "[" et "]") dénotent un envoie de message. En F-Script, ils dénotent un block.

Dans l'exemple, la chaîne de caractère contenant le code F-Script est codée en dur, mais il est bien sûr possible d'utilisé une chaîne construite dynamiquement par programme.

Exemple 2: passage de paramètres et récupération du résultat

Intéressons nous maintenant au passage de paramètres entre Objective-C et F-Script et à la récupération d'un résultat. Comme nous l'avons indiqué, tous se passe avec les Blocks comme en F-Script, même si nous les manipulons maintenant depuis Objective-C. Dans l'exemple suivant, on suppose que l'on dispose d'un NSArray, nommé "airplanes", constitué d'objets de classe Airplane. Ce tableau sera transmis en paramètre au code F-Script. Les objets de classe

Airplane répondent à la méthode "capacity" (qui retourne un int) et "location" (qui retourne une NSString). On souhaite sélectionner dans un nouveau tableau les avions se trouvant actuellement à Chicago et dont la capacité est supérieure ou égale à 200.

```
NSArray *airplanes;
...

NSArray *selectedAirplanes = [[@"[:a| a at:a location =
'CHICAGO' & (a capacity >= 200)]" asBlock] value:airplanes];
```

Exemple 3: object graph

Les blocks F-Script étant des objets, ils peuvent êtres référencés par d'autres objets. Dans cet exemple, un block sera la target d'un NSButton. De plus, des variables F-Scripts, à l'intérieur du code des blocks, peuvent référencer des objets externes. Dans cet exemple, le code F-Script référencera un NSTextField.

Supposons que l'on dispose, dans un programme Objective-C, d'un NSButton et d'un NSTextField. On souhaite utiliser un block F-Script pour afficher la date et l'heure courante dans le text field lorsque le bouton est pressé.

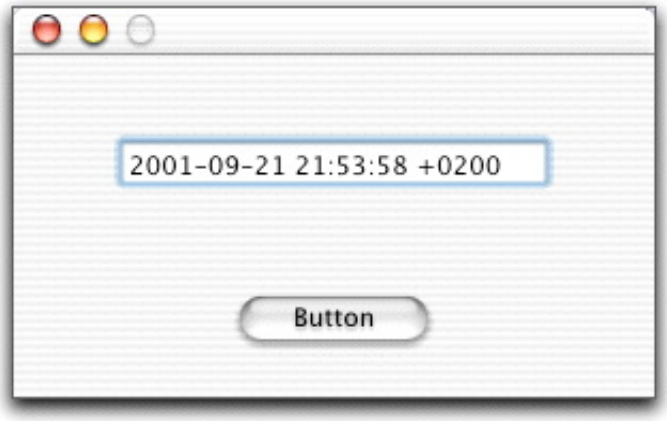

Figure 1. The mini-application we will build

Mais comment établir un lien entre une variable F-Script et le text field externe? Il suffit d'utiliser la technique, assez fréquente avec F-Script, du block pondeur: notre block est pondu par un autre block qui se charge d'établir le lien avec le NSTextField ,qu'on lui passera en argument, en utilisant le fait que les blocks sont des "closures".

Nous donnons ici le programme complet de l'exemple. Les parties importantes sont en couleur.

```
#import <Cocoa/Cocoa.h>
#import <FScript/Fscript.h>

int main(int argc, const char *argv[])
{
  NSApplication *NSApp = [NSApplication sharedApplication];
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSButton *button = [[[NSButton alloc] initWithFrame:NSMakeRect(100,20,100,30)]
                                        autorelease];
```

```
  NSTextField *textField  = [[[NSTextField alloc]
                              initWithFrame:NSMakeRect(50,100,200,20)] autorelease];
  NSWindow *mainWindow = [[NSWindow alloc]
                          initWithContentRect:NSMakeRect(100,100,300,160)
                          styleMask:NSClosableWindowMask | NSTitledWindowMask
                          backing:NSBackingStoreBuffered defer:false];

  Block *pondeur = [[@"[:textField| [textField setStringValue:NSDate now
                        printString]]" asBlock] retain];
  Block *printDate = [[pondeur value:textField] retain];

  [[mainWindow contentView] addSubview:button];
  [[mainWindow contentView] addSubview:textField];
  [button setBezelStyle:1];

  [button setTarget:printDate];
  [button setAction:@selector(value:)];

  [mainWindow orderFront:nil];
  [pool release];
  [NSApp run];
  return 0;
}
```

Gestion des erreurs

Dans les exemples précédents, nous n'avons pas traité la possibilité de survenue d'une erreur. En effet, le code F-Script étant ici entièrement connu, nous savons que sa syntaxe est correcte et qu'il n'existe pas de risque d'erreur F-Script à l'exécution.
Dans des cas plus complexes, des erreurs peuvent survenir à deux niveaux :

- Au moment de la création d'un block à partir d'une NSString. En effet, la syntaxe du code F-Script est validée.
- Au moment de l'exécution du block (par exemple : division par zéro).

Pour gérer le cas d'une erreur de syntaxe, on utilisera la méthode "asBlock:onError:" plutôt que "asBlock" (cf. guide F-Script). Une erreur à l'exécution donne lieu à la levée d'une exception.

Cela dit, l'API FSInterpreter, que nous présentons dans la section suivante, offre une gestion des erreurs plus facile et est donc conseillé dans ce type de situation.

4. Utilisation de la classe FSInterpreter

La technique décrite précédemment est très facile à utiliser et adaptée à de nombreuse situation. Néanmoins, certains cas requièrent plus de contrôle et des fonctionnalités supplémentaires. On se tourne alors vers la classe FSInterpreter, qui offre un contrôle complet de l'interpréteur F-Script. Cette classe assure en particulier :

- Une gestion plus facile des erreurs.
- Le maintien de contexte, c'est-à-dire la possibilité d'exécuter différents bouts de code F-Script, à des moments différent, dans un seul et même workspace.

Une instance de FSInterpreter représente un interpréteur F-Script associé à un workspace. Il est facile à créer en Objective-C :

```
FSInterpreter *myInterpreter = [[FSInterpreter alloc] init];
```

Un interpréteur permet d'ajouter des variables dans le workspace associé, ou de modifier la valeur de variables existantes :

```
[myInterpreter setObject:[NSDate date] forIdentifier:@"myDate"];
```

Un interpréteur permet de consulter la valeur d'une variable dans le workspace :

```
BOOL found;

[myInterpreter objectForIdentifier:@"foo" found:&found];
```

Il est également possible de récupérer la liste des noms de variables définies dans le workspace, sous forme d'un tableau de chaînes de caractère :

```
NSArray *identifiers = [myInterpreter identifiers];
```

Pour exécuter du code F-Script, il suffit de passer une chaîne de caractère contenant ce code à la méthode "execute:" de l'interpréteur. Celle-ci retourne un objet de classe FSInterpreterResult. Voici un exemple avec le "hello world " F-Script :

```
FSInterpreterResult *result = [myInterpreter execute:@"sys
log:'hello world'"];
```

L'objet FSInterpreterResult offre un contrôle total sur le résultat de l'exécution. Les méthodes "isSyntaxError", "isExecutionError" et "isOk", qui retournent un booléen, permettent d'en connaître le statut. En cas d'erreur, les méthodes "errorMessage" et "errorRange" permettent d'obtenir le message d'erreur et l'endroit concerné dans le code source F-Script. La méthode " inspectBlocksInCallStack" permet d'ouvrir l'inspecteur graphique des blocks présents dans la callstack correspondant à l'erreur. En cas d'exécution sans erreur, la méthode "result" permet d'obtenir le résultat de l'évaluation du code F-Script.

Pour illustrer cette API, voici un programme complet, à compiler sous Project Builder comme un « Foundation Tool » et à lancer depuis Project Builder ou une fenêtre de terminal UNIX. Il s'agit d'une interface ligne de commande pour F-Script : le programme li une commande sur son entrée standard, l'exécute, affiche le résultat sur sa sortie standard puis recommence. À noter qu'il s'agit d'un programme non graphique (n'utilisant pas l'Application Kit). Il ne permet donc pas l'utilisation des fonctionnalités graphiques de F-Script.

```
#import <stdio.h>
#import <Foundation/Foundation.h>
#import <FScript/FScript.h>

int main (int argc, char **argv, char **env)
{
  FSInterpreter *interpreter;
  FSInterpreterResult *execResult;
  char c_command[10000];
  NSString *command;

  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  interpreter = [[FSInterpreter alloc] init]; // create the interpreter
  [pool release];
```

```
  while(1)
  {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    command = [NSString  stringWithCString:fgets(c_command, 10000, stdin)];
    execResult = [interpreter execute:command]; // execute the F-Script command

    if ([execResult isOk]) // test status of the result
    {
      id result = [execResult result];

      // print the result
      if (result == nil) puts("nil");
      else               puts([[result printString] cString]);
      if (![result isKindOfClass:[FSVoid class]]) putchar('\n');
    }
    else
    {
      // print an error message
      puts([[NSString stringWithFormat:@"%@ , character %d\n",[execResult
errorMessage],[execResult errorRange].location] cString]);
    }
    [pool release];
  }
  return 0;
}
```

5. Composant FSInterpreterView et Palette pour Interface Builder

La classe FSInterpreterView, sous-classe de NSView, offre un composant graphique interactif F-Script de type command line interface. Le composant possède son propre interpréteur F-Script interne et est prêt à l'emploi. Il peut être instancié et manipulé par programme.

Figure 2. An FSInterpreterView

Cet objet possède des méthodes permettant de modifier la taille de la police, afficher un message de service destiné à l'utilisateur, insérer une commande ou encore récupérer l'objet FSInterpreter associé.
Voici un programme simple qui ouvre une fenêtre contenant un FSInterpreterView.

```
#import <Cocoa/Cocoa.h>
#import <FScript/Fscript.h>

int main(int argc, const char *argv[])
{
```

```
  NSApplication *NSApp    = [NSApplication sharedApplication];
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  // Create a window
  NSWindow *mainWindow  = [[NSWindow alloc]
                           initWithContentRect:NSMakeRect(100,100,500,400)
                           styleMask:NSClosableWindowMask | NSTitledWindowMask
                           backing:NSBackingStoreBuffered defer:false];

  // Create an FSInterpreterView
  FSInterpreterView *fscriptView = [[[FSInterpreterView alloc]
                                     initWithFrame:NSMakeRect(0,0,0,0)]
                                     autorelease];

  // Insert the FSInterpreterView inside the window
  [mainWindow setContentView:fscriptView];

  [fscriptView setFontSize:16]; // We want big fonts !
  [fscriptView notifyUser:@"Welcome in F-Script !"]; // Display a message
  [mainWindow orderFront:nil];
  [pool release];
  [NSApp run];
  return 0;
}
```

Un FSInterpreterView peut également être manipulé directement depuis Interface Builder grâce à la palette FscriptPalette.palette.

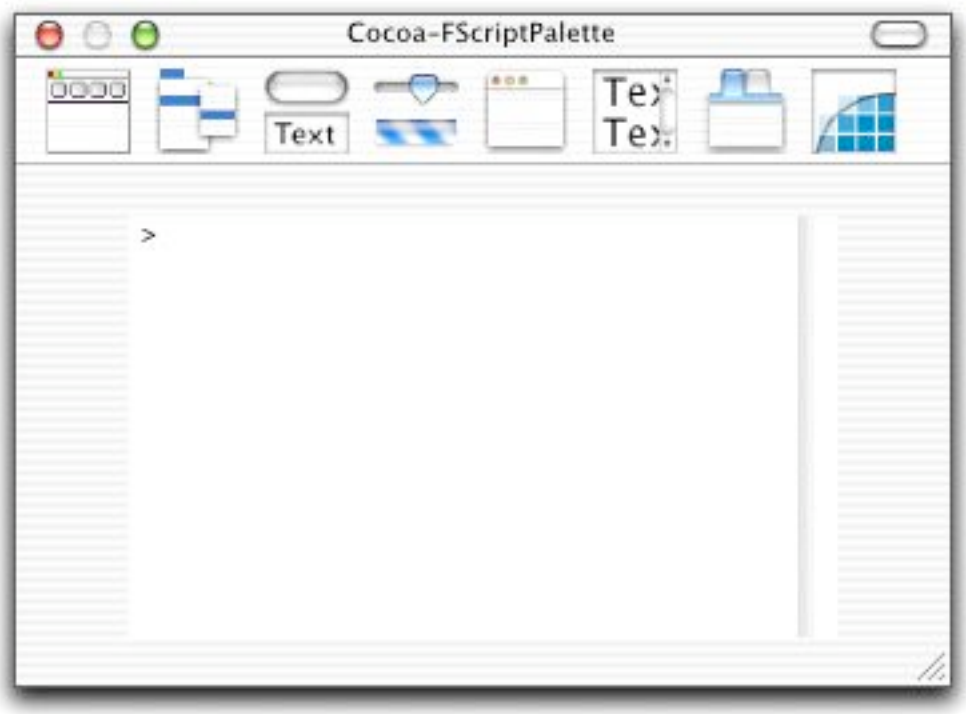

Figure 3. The F-Script palette for Interface Builder

La palette permet de placer un FSInterpreterView dans une interface graphique par simple drag and drop.

Elle offre également une autre fonctionnalité, le « live mode », qui permet d'utiliser F-Script directement depuis Interface Builder (sans entrer en mode test). En live mode, il est possible d'établir des connexions entre les objets d'interface et des objets F-Script (par exemple, établir une connexion entre un bouton et un block) et de sauvegarder tout cela dans la nib file. Grâce à cette fonctionnalité (encore expérimentale au moment de la rédaction de cet article), il est possible de construire une application graphique entièrement depuis Interface Builder, en associant des blocks de code F-Script aux éléments de l'interface.

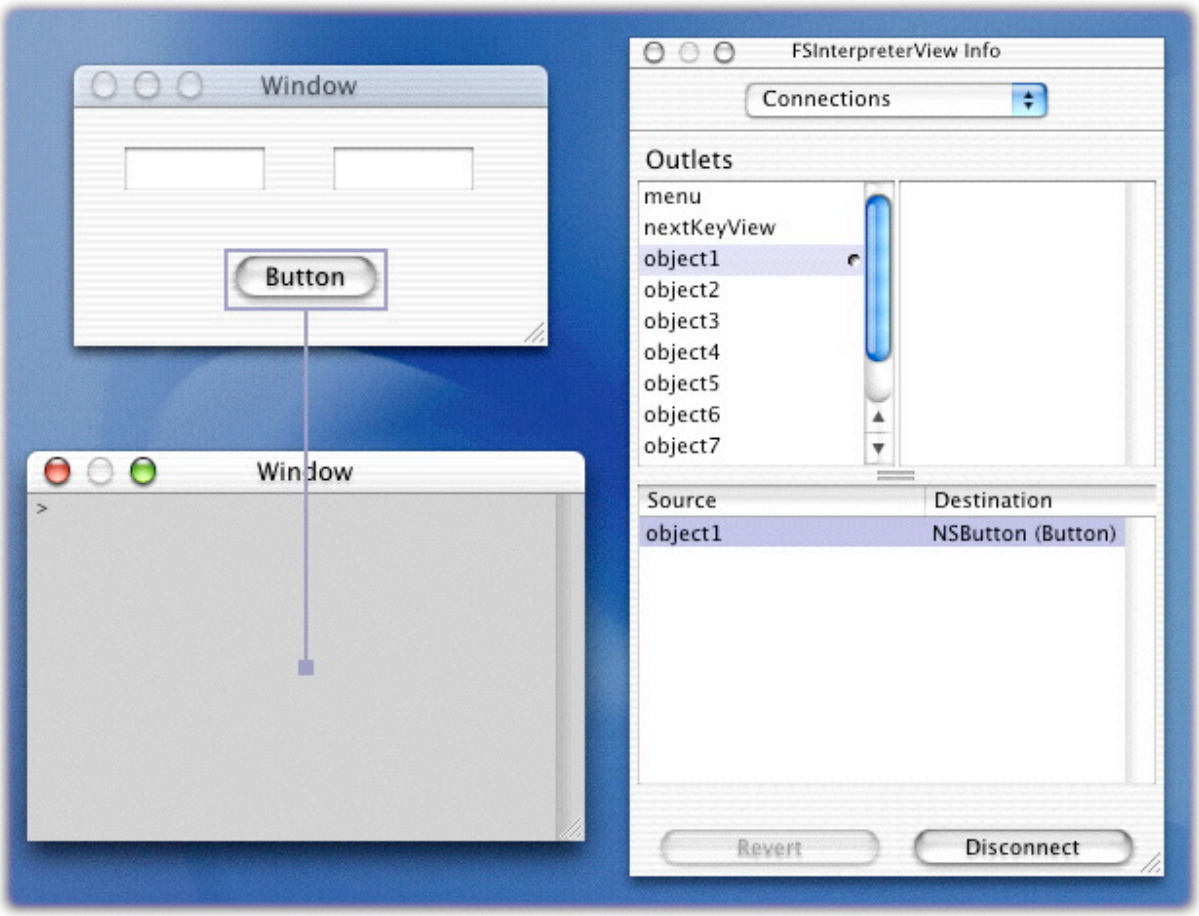

Figure 4. F-Script live mode in Interface Builder: connecting F-Script to a button

Pour utiliser le live mode, il faut tout d'abord instancier un interpréteur F-Script. Pour cela, il suffit d'effectuer un drag-and-drop depuis la palette F-Script vers une fenêtre quelconque. L'objet FSInterpreterView ainsi instancié contient l'interpréteur F-Script permettant le fonctionnement du « live mode ». Pour activer ce mode, on double clique sur l'instance de FSInterpreterView. Une fenêtre F-Script active apparaît alors, depuis laquelle il sera possible de configurer les objets de l'interface (y compris leurs outlets), en utilisant directement des instructions F-Script. L'objet FSInterpreterView est quant à lui utilisé pour établir des connexions depuis l'interpréteur F-Script vers des objets externes (fig. 4).

6. One more thing…

Dans cet article, nous avons montré comment les développeurs Objective-C peuvent intégrer F-Script dans leurs applications. Mais qu'en est-il des applications existantes ? C'est ici qu'interviens F-Script Anywhere. Cet outil, développé par Nicolas Riley et Ken Ferry, est

capable d'injecter dynamiquement un environnement F-Script complet au sein de toute application Cocoa. Cela permet d'explorer et de prendre le contrôle des objets se trouvant au sein de ces applications. F-Script Anywhere est un projet open source disponible sur le site web de Ken Ferry à http://homepage.mac.com/kenferry/software.html#fsa

PyInjector, une application développée par James G. Speth, permet comme F-Script Anywhere d'injecter F-Script dans une application Cocoa.

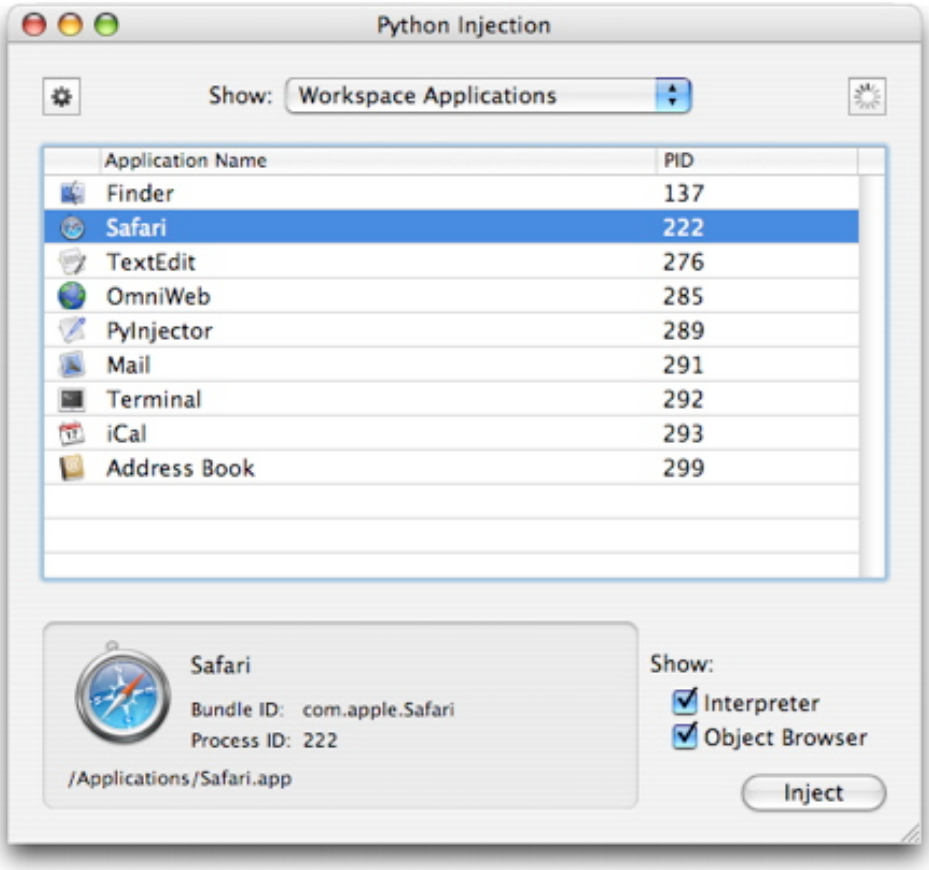

Figure 5. PyInjector permet d'injecter F-Script dans d'autres applications

PyInjector affiche une liste des applications en train de s'exécuter et il suffit de sélectionner dans cette liste l'application dans laquelle on souhaite injecter F-Script.